EUROGRAPHICS 2025 / A. Bousseau and A. Dai
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 2

Approximating Procedural Models of 3D Shapes with Neural Networks

Ishtiaque Hossain[1] I-Chao Shen[2] Oliver van Kaick[1]

[1] Carleton University, Canada
[2] The University of Tokyo, Japan

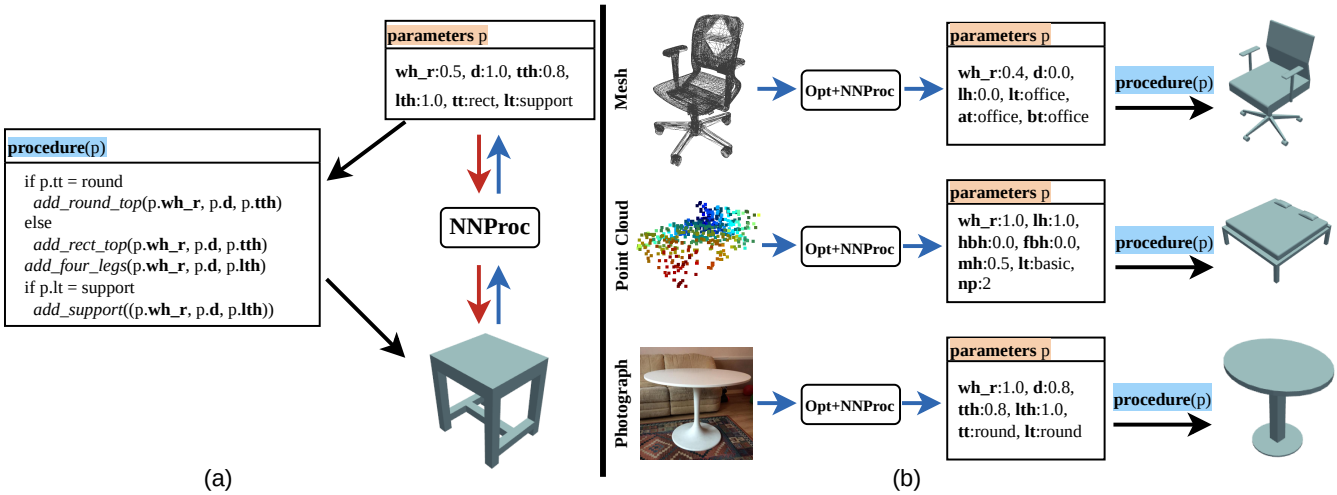

Figure 1: *We introduce a neural network architecture,* NNProc*, which approximates procedural models' forward shape generation (→) and their inverse mapping by aligning latent spaces of parameters and shapes. (a) After training with a given procedural model, NNProc can generate shapes from given parameters (→) and predict parameters for shapes (→). (b) NNProc can also be used in an optimization setting to predict parameters for objects provided in different representations, such as 3D shapes, point clouds, and photographs. Note that we only show a subset of the parameters in the boxes for clarity.*

Abstract

Procedural modeling is a popular technique for 3D content creation and offers a number of advantages over alternative techniques for modeling 3D shapes. However, given a procedural model, predicting the procedural parameters of existing data provided in different modalities can be challenging. This is because the data may be in a different representation than the one generated by the procedural model, and procedural models are usually not invertible, nor are they differentiable. In this paper, we address these limitations and introduce an invertible and differentiable representation for procedural models. We approximate parameterized procedures with a neural network architecture NNProc that learns both the forward and inverse mapping of the procedural model by aligning the latent spaces of shape parameters and shapes. The network is trained in a manner that is agnostic to the inner workings of the procedural model, implying that models implemented in different languages or systems can be used. We demonstrate how the proposed representation can be used for both forward and inverse procedural modeling. Moreover, we show how NNProc can be used in conjunction with optimization for applications such as shape reconstruction from an image or a 3D Gaussian Splatting.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics; Shape modeling;***

1. Introduction

The creation of 3D content, especially 3D objects, is important in several industries, such as game development, movie production, and simulation. Significant time and effort go into the development of 3D shapes as this task requires skilled artists to meticulously create content using an interface. The problem is exacerbated when a large collection of 3D content has to be developed. Thus, there is a growing need for methods that can automatically synthesize 3D objects while minimizing human input. Several techniques have been proposed in the literature to achieve this goal, and the latest research advances include deep neural networks that can learn to represent and then synthesize 3D shapes, for example, using transformers or diffusion models [HPG*22, YLM*22, ZVW*22, HLHF22, LGT*23, ZTNW23]. The downside of these learning-based methods is the need for large amounts of training data that resemble the target shapes. In addition, it is difficult to control the results of the generative models, and the generated shapes are often not readily given in a representation suitable for efficient visualization, such as a triangle mesh.

Another approach for synthesizing large collections of shapes is to use procedural generation. In procedural modeling, there is a set of rules, usually implemented as a procedure or computer program, that can create a shape from input parameters. Procedural modeling can be applied to generate various types of shapes, such as organic, architectural, and man-made objects [STBB14]. One advantage of using procedural modeling is that we can quickly generate a large number of diverse shapes by varying the input parameters. More importantly, unlike generative models, shapes can be easily edited by changing their parameters, and the procedural model can be designed to ensure that the resulting shapes are free of artifacts and self-intersections. The main requirement for using procedural generation is the creation of the procedure by a skilled developer with an artistic eye.

Nevertheless, using a predefined procedural model can still be challenging even for skilled users because they need to explore a large parameter space to create the shape that they desire. Methods have been proposed to predict parameters for input shapes, which can be used to replicate a given shape or to provide a starting set of parameters for further exploration [SPK*14, HKYM17]. However, most procedural models are equivalent to a non-differentiable computer program, which implies that efficient gradient-based optimization methods cannot be used to estimate parameters. Some recent work defines procedural models in terms of some Domain Specific Language (DSL) which in turn allows defining gradients at arbitrary locations of the shapes. These solutions include systems where the user can edit specific parts of the generated shapes [LSL*19, MB21, CSQ*22], or where optimization methods such as automatic differentiation are used [GKG*22], which require the procedural model to use predefined differentiable operations. These methods are suitable solutions for parameter prediction, but are constrained by the use of specific DSLs or operations.

In this paper, we propose *NNProc*, a neural network architecture for approximating procedural models of shapes (Figure 1). Our method can approximate existing procedural models encoded as programs, node graphs, or other representations. Our key idea is to approximate a procedural model and *its inverse mapping* with a differentiable neural network, which is learned from data generated by the procedural model. Then, inverse tasks that are difficult to accomplish with the original procedural model can be carried out more easily with the neural network approximation.

We note that a procedural model can be approximated as a neural network that generates shapes from input parameters [NGDA*16, HKYM17]. However, such a design does not impose a meaningful structure on the latent space of the network, which still leads to the failure of inverse optimization tasks. Instead, we propose to learn both the forward and inverse mappings of the procedural model by aligning latent spaces for parameters and shapes. The different pieces of the trained network can then be used to perform various tasks, including the prediction of procedural parameters from 3D shapes. Moreover, since the individual components of the network are differentiable, gradient based optimization algorithms can also be applied to any portions of the network for optimization-based tasks, such as predicting parameters from images of shapes. Finally, the neural network approximation is agnostic to how the procedural model is implemented, as we train the network with training data automatically collected from the procedural model.

We demonstrate that our network can approximate different classes of man-made objects represented by different procedural models (primitive-based and node-based procedural models). It can generate shapes from parameters (forward procedural modeling) for fast visualization. More importantly, the proposed network can be used to predict parameters of the procedural model for replicating shapes. To demonstrate the inverse mapping, we present results from experiments where we combine our network with iterative optimization to infer shape parameters from 3D shapes, silhouette images, photographs, and views sampled from a 3D Gaussian Splatting. To summarize, our main contributions are:

- We introduce a generalized representation for procedural models that is both invertible and differentiable since it is based on a simple but versatile neural network.
- We demonstrate that our architecture is applicable to various procedural models implemented in different manners, and can be used for both forward and inverse problems given that it learns latent spaces with meaningful structures.
- By reconstructing 3D shapes from images, we show that our differentiable representation can also be used in optimization-based tasks for inverse mapping.

2. Related work

Procedural modeling can be used to generate a variety of geometry, including man-made objects, organic shapes, terrains, and architectural shapes. Smelik et al. [STBB14] summarize a number of procedural modeling systems for a wide range of types of shapes. In many cases, the procedural model can be viewed as a grammar or a set of rules. Some of these models define the grammar using geometric constraints [MM11, KMG*21], while others use high-level specifications [TLL*11]. Lipp et al. [LWW08] and Patow [Pat12] also investigated methods that enable visual editing of the grammar. The primary focus of all of these methods is shape generation. In contrast, we also consider other related tasks such as parameter inference for given shapes. Thus, we discuss related previous work as follows.

Parameter Inference: Assuming that a procedural model is given, one group of works in the literature propose methods for discovering the parameters that reproduce input shapes, which is one task related to *inverse procedural modeling*. Stava et al. [SPK*14] introduce a procedural model of trees and propose to use simulated annealing to find optimal parameters for given shapes of trees. Yuan et al. [YBP*24] proposed using differentiable rendering in order to infer parameters of a given procedure for a given target 3D shape. Parameter prediction from sketches has also been investigated for shape modeling [NGDA*16, HKYM17]. Additionally, parameter prediction can also be performed for objects other than 3D shapes, such as knitwear and texture [TMK*19, HDR19]. However, the methods proposed in this body of work only address the issue of inferring parameters for a given input and are not well suited for being plugged into a setup where parameters can be optimized for various representations of shapes.

Instead of performing parameter prediction for given shapes, a few works let the user directly edit shapes generated by the procedural models and then transfer back the changes to the parameters. For example, Lipp et al. [LSL*19] propose a method for discovering good local edit operations in procedurally created shapes, which have a known mapping to the model's parameters. Michel et al. [MB21] and Pearl et al. [PLH*22] propose systems that propagate edit operations to the parameters of procedural models, while Cascaval et al. [CSQ*22] introduce a bidirectional editing system for CAD models. On the other hand, Yumer [YAMK15] proposes to circumvent the parameter prediction by instead exploring an embedding space linked to the parameter space. Although these methods allow to connect parameters to shapes more efficiently, the representations are quite specialized to the procedural models at hand, and the user is not able to provide a shape in some other modality as a starting point for editing. In comparison, our method enables a fully differentiable representation of procedural models that is not specific to one application.

Procedure Inference: Beyond the prediction of shape parameters, another body of work proposes methods to automatically derive the procedural model itself from given shapes. Šťava et al. [ŠBM*10] and Guo et al. [GJB*20] investigated automatic inference of L-systems for organic shapes. Wu et al. [WYD*14] and Nishida et al. [NBA18] proposed different techniques to automatically infer a grammar for building facades. Demir et al. [DAB14, DAB16] also explored this direction, but for architectural shapes. Bokeloh et al. [BWS10] inferred the grammar by discovering partial symmetries in the shapes, while Guérin et al. [GDGP16] adopted a sparse representation of terrains. Merrell [Mer23] proposed a method that infers graph-grammars from example shapes drawn from both the 2D and the 3D domain. Mathur et al. [MPZ20] proposed a system where procedures for CAD programs are generated via user edits on an interface. In the domain of materials, Guerrero et al. [GHS*22] developed a transformer based method for generating node-graphs for materials. Hu et al. [HGH*23] also proposed a node-graph generation technique based on text or image prompts, while Hu et al. [HHD*22] proposed an inverse procedural pipeline based on a hierarchical decomposition of an input material.

Another group of work takes a different approach to inverse procedural modeling, representing shapes as individual programs and framing the problem of inverse procedural modeling as a program synthesis problem. Sharma et al. [SGL*22] proposed a method that predicts Constructive Solid Geometry programs for 2D and 3D shapes. In many cases, the programs are written following some Domain Specific Language (DSL). Tian et al. [TLS*19] proposed such a DSL, along with a method that learns both to generate and execute programs. Jones et al. [JBX*20] also put forth their own DSL and program-synthesis method. In subsequent studies, Jones et al. [JCG*21, JGMR23] extended their work by proposing techniques to discover macros and abstractions from shape programs. In another work, Jones et al. [JWR22] proposed a generalized framework for learning to generate DSL-based programs.

Given that inferring procedures is a challenging and often ill-posed problem, the approaches discussed above tend to be quite specialized to the DSLs involved and still require parameter inference when fitting the models to input data. In contrast, in our work, we focus on the problem of parameter inference, which is of importance when detailed procedural models with a large number of parameters are already available and intended for reproducing data given in different modalities.

Differentiable Procedural Modeling: A relatively smaller number of works explore the idea of using a differentiable procedural model and/or its inverse to perform tasks other than shape generation. Shi et al. [SLH*20] and Hu et al. [HGH*22] use differentiable building blocks in their node-graphs for material design, so that the entire procedural models are also differentiable. Gaillard et al. [GKG*22] propose a differentiable procedural model that facilitates shape editing. The method creates a proxy differentiable representation of a procedural model encoded as a node graph, but requires that the nodes be differentiable operations. Although our method is inspired by elements from these studies, instead of requiring differentiable building blocks or modifying a given procedural model into a differentiable form, our main idea is to train a neural network to approximate the model, which allows us to be agnostic to the implementation of the model and learn both the forward and inverse mappings of the model. This makes the approximation applicable to a wide range of use cases.

Shape Inversion and Editing: Some recent studies explore the idea of performing shape inversion and exploration with generative models of shapes. For example, Zeng et al. [ZVW*22] propose a diffusion-based method for point cloud generation that enables conditional shape synthesis and exploration of the latent space. Hu et al. [HHL*24] introduce a wavelet-domain diffusion model for shape generation, which also allows to regenerate shape parts. Lin et al. [LMW*22] put forth an adaptive overfitting technique for neural representations of shapes that facilitate shape synthesis and shape editing. Shen et al. [SSW*24] establish forward and backward mappings between an image latent space and a shape latent space to enable shape editing in a generated image. Kusupati et al. [KGTK24] fit a parametric implicit surface template to an input mesh in order to enable editing of the input mesh via the parameters of the proxy geometry. These methods enable shape generation from latent codes or editing. However, no procedural model is linked to these types of models. We make use of some of these ideas in our work and map shapes and parameters to latent spaces which are then aligned to each other.

3. Approximation of procedural models

Overview. Given a procedural model PM that takes a parameter vector as input and produces a shape as output, our goal is to approximate PM as a differentiable function. To be more specific, the procedural model PM establishes a mapping from the parameter space P to the shape space S, where $\text{PM}(p) = s$, with $p \in P$ and $s \in S$. Usually, the mapping is unidirectional and such a procedural model can only be used for shape generation. However, when we are interested in the inverse mapping $\text{PM}^{-1}(s) = p$, with $s \in S$ and $p \in P$, i.e., estimating the parameters for given shapes, this cannot be obtained directly from the procedural model. One solution is to pose the inverse mapping as an optimization problem and search for the optimal parameters for given shapes. However, in most cases, the procedural model is not differentiable, and thus efficient gradient-based optimization methods cannot be used.

To overcome this challenge, we aim to approximate the procedural model as a differentiable function PM_d. We achieve this by representing PM_d as a neural network which can generate shapes from input parameters based on a learned latent space. Moreover, to achieve inverse mapping, we similarly represent it with a neural network encoding a latent space. Then, we combine both mappings together in a single network and make the mappings consistent by aligning the latent spaces for parameter and shape prediction. The result is a network that approximately recovers both the forward and backward mappings of the procedural model. We train the network in a manner that is agnostic to the inner workings of the procedural model by automatically generating training data with the procedural model. We describe our assumptions on the procedural model, the network architecture, and training procedure as follows.

3.1. Procedural Model Assumptions

Since our goal is to represent a procedural model PM with a new differentiable form PM_d, and our method is agnostic to the implementation of the model, we only have assumptions on the input of the model, i.e., the types of parameters that can be provided. Specifically, we assume that the procedural model takes a vector of parameters as input and creates a shape. We also assume that the vector of input parameters to PM can have a mix of four different kinds of parameters:

- A parameter can be a *continuous scalar*, e.g., a real value that encodes the width of a shelf.
- A parameter can be an *integer value*, e.g., one that represents how many columns a shelf contains.
- A parameter can be an *item from a set*, e.g., one that indicates which type of leg a shelf has, out of three possible types.
- A parameter can be a *binary value*, e.g., to represent whether the back of a shelf is open or not.

3.2. Network Architecture

The neural network for approximating the forward and inverse mappings of the procedural model follows an architecture similar to that of an Autoencoder. Figure 2 shows the overall architecture. The network has separate encoders that encode the parameters (PE) and the shapes (SE) into latent vectors z_p and z_s. The network is trained so that the latent spaces learned by the two encoders are aligned to each other, based on the loss function described in Section 3.4. Thus, in practice, the two decoders decode latent vectors from a common latent space into parameters and shapes.

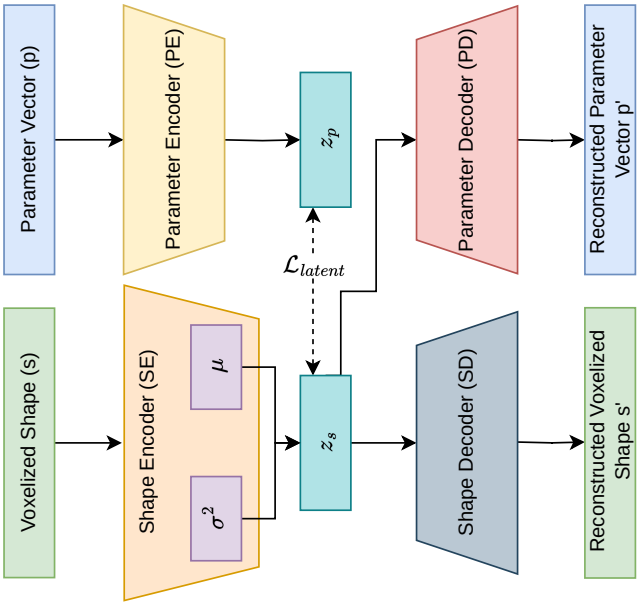

Figure 2: *Architecture of our neural network NNProc that approximates the forward and inverse mappings of a procedural model. The network is composed of encoders and decoders for parameter vectors and 3D shapes, which work on the latent spaces z_s and z_p aligned to each other by $\mathcal{L}_{latent}$. Please refer to the text for details.*

Note that the procedural model can use any encoding technique for the output shapes, such as meshes, voxel-grids or Signed Distance Fields (SDF). Each technique comes with its own advantages and limitations. For example, meshes can represent high quality shapes, but their unstructured nature leads to more complex network architectures. Voxel-grids are structured data but a good approximation of fine details requires high resolution voxel-grids, which are computationally expensive. SDFs are resolution independent, but they have to be represented either in an unstructured form or as samples on a grid. For our network, we opt to use voxel-grids since other shape representations can be easily transformed into this format for encoding and grids are good enough for previewing shapes. Users can then run the original procedural model with predicted parameters to obtain the final shape.

The encoder and the decoder for parameters are Multi-Layer Perceptrons (MLPs), while the encoder and the decoder for shapes are Convolutional Neural Networks (CNNs). Moreover, both of our decoders mirror the corresponding encoders in terms of how the layers are arranged. The parameter encoder has three fully-connected ReLU-activated layers with 256, 256, and 128 neurons. The shape encoder consists of five convolutional ReLU-activated layers with 8, 16, 24, 32, and 40 channels, followed by two fully connected layers with 256 and 128 neurons. The continuous and binary values in the input parameter vector are normalized to $[0,1]$ and $\{0,1\}$,

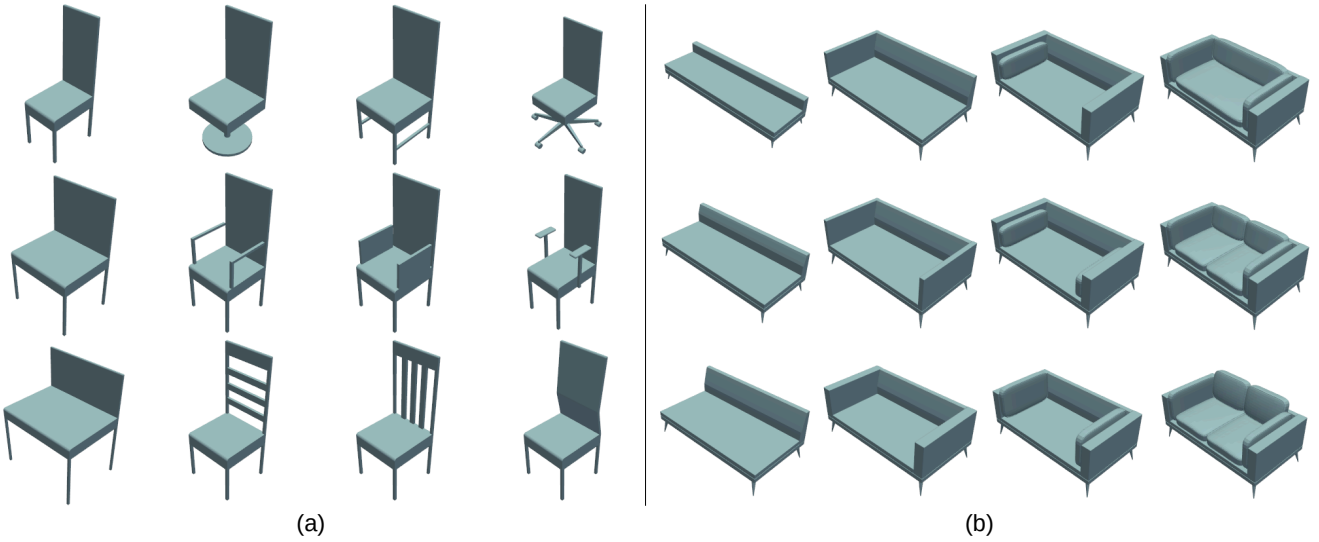

Figure 3: *Examples of a variety of (a) chairs created by the primitive-based procedural model and (b) sofas obtained with the node graph-based procedural model.*

respectively, and the categorical values are one-hot-encoded. The output layers corresponding to continuous and binary values are sigmoid-activated. This approach also ensures that the predicted parameters are always valid. The input voxel values are in $\{0,1\}$ and the last layer in the shape decoder is also sigmoid-activated.

3.3. Training Data

The training dataset $T = \{P', S' = PM(P')\}$ consists of shapes paired with their corresponding parameter vectors. Training data is generated by first obtaining P', which is a randomly-sampled subset of P, and then invoking the procedure on the sampled parameter vectors to obtain the corresponding subset of shapes S'. As an example, assuming the parameter vector has 3 elements and they are sampled into 6, 3 and 5 steps, we sample $6 \times 3 \times 5$ possible vectors. In addition, all scalar parameters are not necessarily sampled into equal number of steps. For instance, it may suffice to sample the thickness of the tabletop into 3 steps, but more samples are needed for the width-to-height ratio of the table. In our experiments, the number of steps used to sample scalar parameters vary between 3 and 8. The binary parameters are sampled into 2 steps. The integer and set element parameters are sampled into a number of steps determined from their allowable range and set-size respectively. The output shapes generated by the procedural models are represented as triangle meshes. However, our network expects the shapes to be represented as voxels, which is why the shapes are then voxelized into $64 \times 64 \times 64$ voxel-grids for encoding by the network.

3.4. Loss Function

We design the loss function so that the network learns a common latent space for parameter and shape prediction while also following the principles of Variational Autoencoders (VAE), i.e., the latent vectors come from a normal distribution, as this regularizes the latent space and enforces it to be more continuous.

The loss function used for optimizing the network is defined as:

$$\mathcal{L} = \mathcal{L}_{param} + \mathcal{L}_{shape} + \mathcal{L}_{KL} + \mathcal{L}_{sim} + \mathcal{L}_{latent},$$

where $\mathcal{L}_{param}$ is the loss between the original parameter vector and the reconstructed parameter vector, $\mathcal{L}_{shape}$ is the loss between the original shape and the reconstructed shape, $\mathcal{L}_{KL}$ is the KL divergence for a standard Gaussian distribution, $\mathcal{L}_{sim}$ is the loss incurred by similar parameters being encoded to dissimilar vectors in the latent space, and $\mathcal{L}_{latent}$ is the loss between the latent vector encoded by the parameter vector (z_p) and the mean encoded by the shape encoder (μ). Thus, $\mathcal{L}_{param}$ and $\mathcal{L}_{shape}$ ensure that the network predicts the correct output for the forward and inverse mappings based on the training data, $\mathcal{L}_{KL}$ regularizes the latent space, $\mathcal{L}_{sim}$ further regularizes the latent space by bringing latent vectors of similar parameter vectors close together, and $\mathcal{L}_{latent}$ aligns the two latent spaces to each other.

Specifically, $\mathcal{L}_{param} = \sum_{s \in S'} \sum_{i=1}^{n} L_i^s$, where L_i^s represents the loss for the i-th element of the parameter vector of shape s, and n is the size of the parameter vector. For scalar parameters, L_i is a squared-difference loss and for other kinds of parameters, L_i is a cross-entropy loss. $\mathcal{L}_{shape} = \sum_{s \in S'} L_{CE}(s, s_{recon})$, where $L_{CE}(s, s_{recon})$ represents the cross-entropy loss between shape s and its reconstruction s_{recon}. $\mathcal{L}_{KL} = -\frac{1}{2} \sum_{s \in S'} (1 + \log \sigma_s^2 - \mu_s^2 - e^{\log \sigma_s^2})$, where μ_s and σ_s^2 are the mean and variance for shape s respectively, encoded by the shape encoder. $\mathcal{L}_{sim} = \sum_{s_1, s_2 \in S'} \|CosSim(\mu_1, \mu_2) - CosSim(p_1, p_2)\|^2$, where μ_1, μ_2 are encoded means for shapes s_1, s_2, the parameter vectors corresponding to the shapes are p_1, p_2, and $CosSim$ is the cosine similarity between two vectors. $\mathcal{L}_{latent} = \sum_{s \in S'} \left\|\mu_s - z_{p_s}\right\|^2$, where z_{p_s} is the latent vector encoded by the parameter encoder for shape s.

4. Results

One of the advantages of our approach is that the network approximation of the procedural model can be used for various tasks. As follows, we first provide details on the procedural models used in our experiments and then explore examples of possible tasks.

4.1. Procedural Models and Training Setup

We test our method on two procedural model implementations that generate five classes of 3D man-made furniture shapes, specifically, beds, chairs, shelves, tables, and sofas. Both models are similar in the sense that they take a vector of parameters as input and create a shape represented as a triangle mesh as output. However, the design and implementation of the two procedural models and the sizes of the input parameter vectors are different.

The first procedural model used in our experiments is a primitive-based procedural model. The model assembles primitives such as cubes and cylinders of different scales and orientations in order to create more complex shapes. The procedural model utilizes the internal data structures used by Blender to create and manipulate the primitives. The transformations applied to the primitives are governed by the parameters to the procedure. The procedural model primarily uses the cube and the cylinder primitives, with scaling, rotation and translation applied to them. In some cases, geometry-modifying operations such as edge bevelling and face extrusion are applied as well. The model is composed of 894 lines of Python code in total (excluding the code of the Blender libraries). Figure 3(a) shows some examples from the chair class that were created using this procedural model. More details about the procedural model are provided in the supplementary material and the code can be found at https://github.com/ihossain-cu/ProcShapes

The second procedural model is a Blender geometry node-based model [Bas21], which creates shapes from the sofa class. The node graph used by the model consists of 830 individual nodes. The model comes with 19 adjustable parameters. However, allowing some of the parameters, especially the ones that control leg-related attributes, to assume arbitrary values can result in implausible geometry. We opt to set 7 such parameters to constant values and allow the remaining 12 to vary. Figure 3(b) shows some examples from the sofa class generated with this model.

For training-data, we use $3,000$ example shapes from each category of shapes generated by the primitive-based procedural model. The geometry-node-based procedural model has more parameters and with our approach for sampling parameter vectors, the number of possible parameter vectors is also very large. In this case, we limit the number of training samples to $10,000$ in order to keep the training time within acceptable range. The primitive-based procedural model takes approximately 4ms on average to generate a shape, while the geometry-node-based procedural model takes 160ms on average. Thus, generating the training data takes approximately 28 minutes in total. We train a separate neural network for each category of shapes, and use a held-out set of 300 example shapes to test each network. It takes approximately 8 hours to train all 5 models on an NVIDIA RTX3090 GPU.

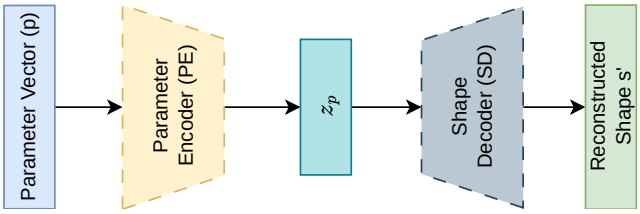

Figure 4: *Network components involved in shape generation. The dashed lines indicate that the parameters for the corresponding component are frozen.*

4.2. Applications of NNProc

We demonstrate in this section how our network can be used in various configurations to perform different tasks. The tasks can be broadly divided into two categories, shape generation (forward procedural modeling) and shape reconstruction. Shapes can be reconstructed by first predicting parameters for given shapes (inverse procedural modeling) and then using the procedural model to replicate the shapes from predicted parameters. We also show that there is more than one approach that can be taken with our network for predicting shape parameters. Parameters can be predicted directly by means of inference, or via optimization. The latter approach allows us to use our network in various other applications, for instance, parameter prediction from images. As follows, we show how our network can be used to perform the following tasks:

- Task 1: Generating 3D shapes from given parameters.
- Task 2: Reconstructing 3D shapes using parameters predicted from 3D shapes via inference.
- Task 3: Reconstructing 3D shapes using parameters predicted from 3D shapes via optimization.
- Task 4: Reconstructing 3D shapes using parameters predicted from images via optimization.

We also compare the results of our parameter-prediction techniques with a baseline method based on optimization. Specifically, we use the COBYQA method available in the scipy package in Python, which is a derivative-free method that minimizes given objective functions [Rag22, RZ24]. For the objective function, we use the Chamfer distance between the original shapes and the reconstructed shapes. We opt to use this baseline in particular because the procedural models in their original forms are not differentiable, and this method is the best-performing optimization method among all the methods available in the package. We would like to point out that our method is GPU-accelerated while the baseline method is not and it may be challenging to adapt the currently available implementation of the baseline method to take advantage of specialized hardware. However, the main goal of this comparison is to provide a general idea of the performance of the methods.

4.2.1. Task 1: Shape Generation

We evaluate how well the neural-network approximation of the procedural model (NNProc) works when the task is to generate shapes from input parameters. As illustrated in Figure 4, for this task, once the original network is trained, we encode the input parameters of the test shapes into latent vectors using the parameter

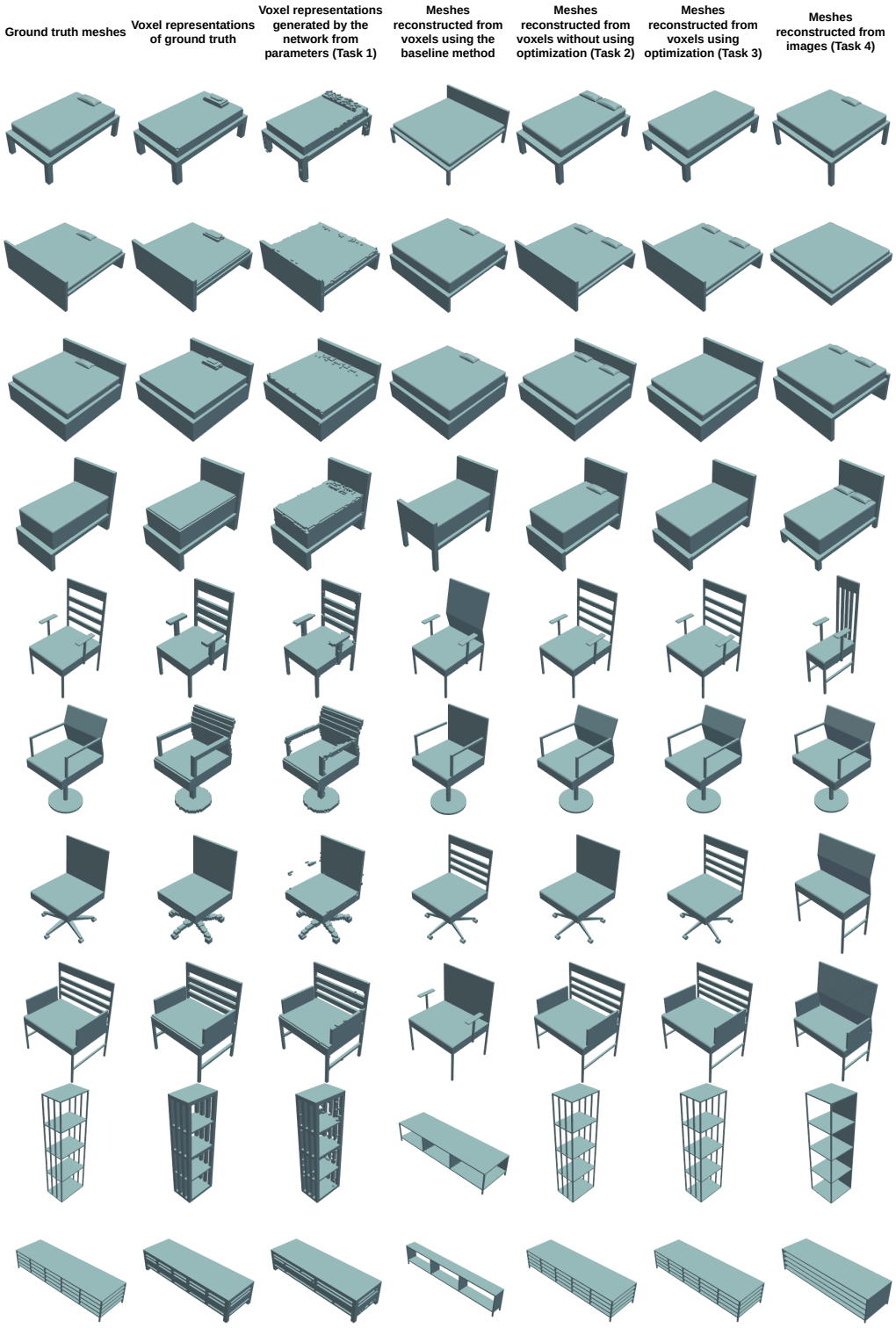

Figure 5: *Results of the various tasks enabled by our neural approximation of procedural models (NNProc) on selected test shapes, including approximating the procedural model (Column 3) and tasks related to inverse mapping for reconstruction (Columns 5-7). Please refer to Sections 4.2.1-4.2.4 for details.*

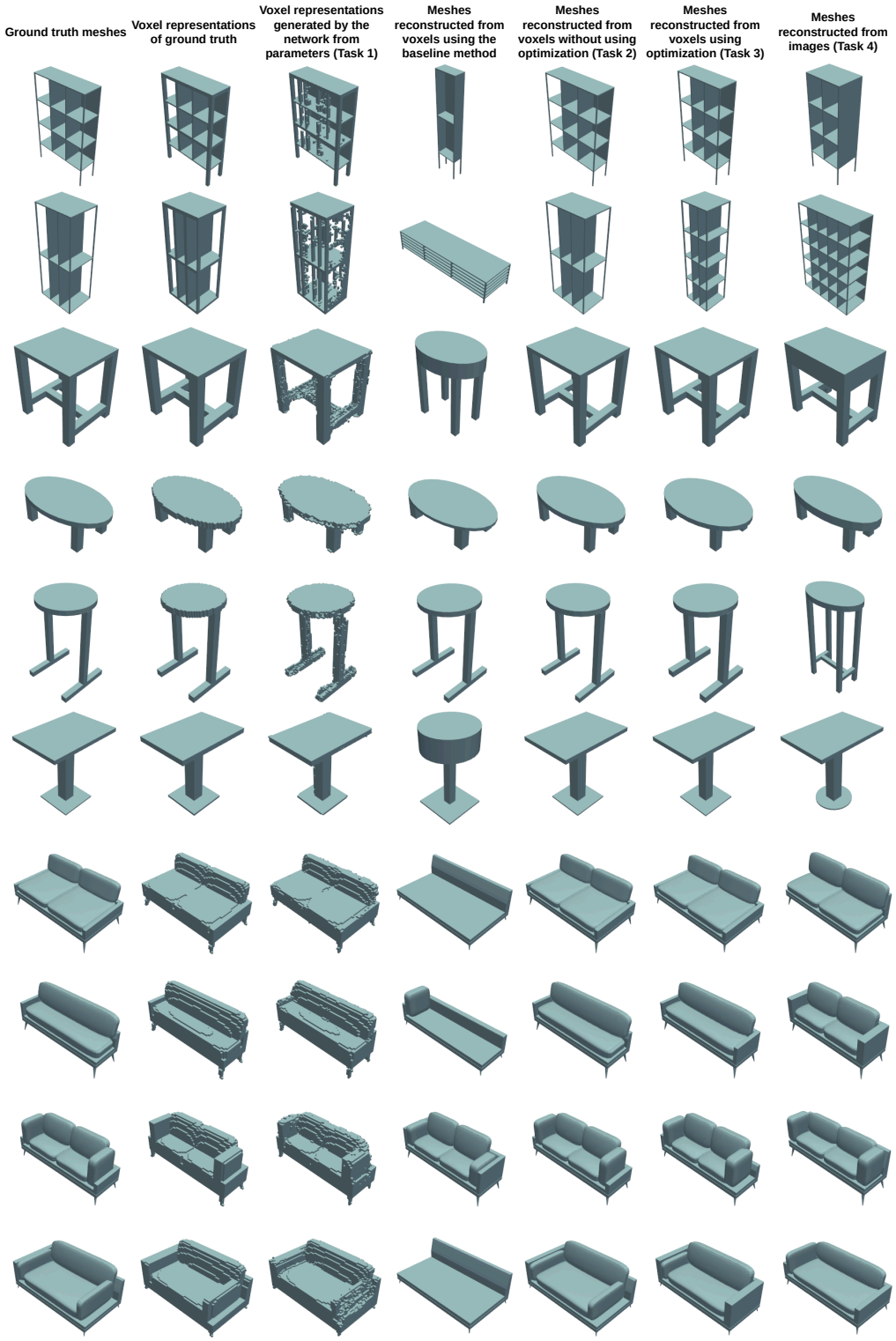

Figure 6: *Results of the various tasks enabled by our neural approximation of procedural models (NNProc) on selected test shapes, including approximating the procedural model (Column 3) and tasks related to inverse mapping for reconstruction (Columns 5-7). Please refer to Sections 4.2.1-4.2.4 for details.*

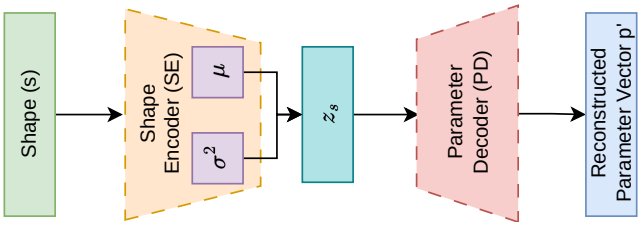

Figure 7: *Network components involved in parameter prediction.*

encoder PE and then use the shape decoder SD to decode shapes from the latent vectors. Formally, given some parameter p, we calculate $\mathrm{SD}(\mathrm{PE}(p))$. With this approach, on average, it takes approximately 1ms to create one shape from its corresponding parameter vector. Figures 5 and 6 show selected examples of generated shapes for qualitative evaluation of the results. The first column shows the original shapes in triangle mesh format and the second column shows their corresponding voxel representations. The third column shows the shapes generated in voxel format from their corresponding parameters using the network.

To provide a quantitative evaluation, we compare the shapes generated by the network with the voxelized representations of the original shapes. The dissimilarity between an original shape and a reconstructed shape is measured by the metric $D_{\mathrm{voxel}} = \frac{100}{R^3} \sum s \oplus s'$, where $s \oplus s'$ represents the logical XOR between the voxels s and s' from each shape and R is the size of the voxel-grid. This metric reports the percentage of erroneous occupancy with respect to the total size of the voxel-grid. The fifth column of Table 1 shows the average dissimilarity between the voxel representations of the test shapes and the voxel representations generated from their corresponding parameters. For reference, the third column shows the average dissimilarity between voxel representations over all pairs of shapes from the same class. We see that the dissimilarity between the original shapes and the reconstructed shapes is less than 1.1%. The visual results are not perfect and have small-scale artifacts, for instance, the network struggles to generate thin structures like the inner walls of the shelves accurately at this resolution. However, the results are sufficient for fast visualization and for verifying that the model is learning to approximate the procedural model.

4.2.2. Task 2: Direct Parameter Prediction

Next, we test how well NNProc predicts parameters for given shapes. In this task, we predict the parameters for voxel representations of unseen shapes and use the original procedural model to reconstruct the shapes as meshes. We encode the test shapes into latent vectors using the shape encoder SE from the trained network and then decode the latent vectors into parameters using the parameter decoder PD, as illustrated in Figure 7. Formally, given a shape s, we calculate $\mathrm{PM}(\mathrm{PD}(\mathrm{SE}(s)))$. The experiment is performed on the same set of test shapes used in the experiments for the previous task. The fifth columns in Figures 5 and 6 show qualitative examples of shapes reconstructed using the network, whereas the fourth columns show examples of shapes reconstructed using the baseline method. To compare an output mesh with the ground truth mesh, we compute the Chamfer distance (D_{chamfer}) between the shapes, where the shapes are first normalized into a unit cube. We report the average distances for all the shapes in the test set in the second column of Table 1.

We see in the visual examples that the reconstructions are quite close to the original shapes, differing only in a few fine details such as the number of pillows on the beds, or the fine details on the surface of the couches. The average Chamfer distances for all the categories are at most 0.0014. For reference, the second column shows the average dissimilarity between mesh representations over all pairs of shapes from the same class. Table 1 also shows that our method significantly outperforms the baseline method in terms of reconstructing shapes faithfully. Our method is also considerably faster than the baseline method, with an average inference time of 0.6ms per shape. For the baseline method, the average time taken for parameter prediction varies between 1s and 18s per shape on an Intel i9-10900KF CPU, depending on a number of factors, such as the number of parameters and the number of triangles.

To show that the alignment of latent spaces in our network does not interfere with the quality of the inverse mapping, we also compare our results with those of a simplified network that only learns the mapping from shapes to parameters. In other words, the network consists of a shape encoder and a parameter decoder. Note that, in contrast to NNProc, such a network is neither suitable for shape generation nor can it be used to optimize parameters for unseen shapes. However, it provides a reference for the inverse mapping. Table 1 shows that the results of the simple network are comparable in quality to the results of NNProc. The inference time is also comparable.

4.2.3. Task 3: Parameter Prediction via Optimization

The approach taken in Section 4.2.2 for parameter prediction is based on the inverse mapping learned by the network. Thus, it is inference-based and does not involve any optimization. The same task can also be performed using an optimization-based approach, which is possible since the components of the network are differentiable. The optimization mainly requires the shape decoder SD from the original network and the parameter decoder PD at the end of the optimization. This approach is relevant in situations where direct prediction via the shape encoder is not possible, but only indirect prediction, e.g., when other modalities of data are used.

In this approach, we optimize μ and σ^2 for the voxel representation of each unseen shape. Then, we sample a latent vector from the distribution (μ, σ^2), which is input into the trained shape decoder. Figure 8 illustrates how the network is used in this optimization task. During the optimization, we calculate the loss between the predicted shape and the target shape, and then back-propagate the error to update μ and σ^2 while keeping the decoder parameters frozen. The values of μ and σ^2 are initially set to 0 and 1, respectively. Formally, given a shape s, we find $\mu^*, {\sigma^2}^* = \arg\min_{\mu,\sigma} L_{CE}(\mathrm{SD}(z_s \sim \mathcal{N}(\mu, \sigma^2), s)$. We observe that, with this approach, μ^* is a local minimum of the reconstruction error. Then, we can generate a shape using the decoded parameters with the original procedural model, i.e., $\mathrm{PM}(\mathrm{PD}(\mu^*))$.

The sixth columns in Figures 5 and 6 show visual examples of test shapes reconstructed with this approach. We observe in the vi-

Table 1: *Quantitative evaluation of the various tasks enabled by our network (Ours) on a test set, along with a comparison to a baseline method based on optimization (Baseline), a simple inverse mapping network (Simple), and an ablation study on the regularization of the latent space with various loss terms. We highlight the best result for each shape class and task.*

Shape class	Avg dissimilarity among meshes D_{chamfer}	Avg dissimilarity among voxels D_{voxel}	Method	Task 1 (D_{voxel})	Task 2 (D_{chamfer})	Task 3 (D_{chamfer})	Task 4 (D_{chamfer})
Table	0.0658	6.38	Baseline		0.0209		
			Simple		0.0003		
			Ours	0.75	0.0003	0.0031	0.0117
			w/o $\mathcal{L}_{KL}$	0.65	0.0007	0.0199	0.0257
			w/o $\mathcal{L}_{sim}$	1.09	0.0003	0.0158	0.0144
			w/o $\mathcal{L}_{KL}$ & $\mathcal{L}_{sim}$	0.63	0.0007	0.0185	0.0318
Sofa	0.0033	5.98	Baseline		0.0013		
			Simple		0.0002		
			Ours	0.34	0.0002	0.0005	0.0016
			w/o $\mathcal{L}_{KL}$	0.51	0.0003	0.0018	0.0028
			w/o $\mathcal{L}_{sim}$	0.49	0.0002	0.0006	0.0016
			w/o $\mathcal{L}_{KL}$ & $\mathcal{L}_{sim}$	0.28	0.0002	0.0021	0.0028
Chair	0.0191	3.85	Baseline		0.0048		
			Simple		0.0001		
			Ours	0.08	0.0001	0.0007	0.0057
			w/o $\mathcal{L}_{KL}$	0.07	0.0003	0.0061	0.0125
			w/o $\mathcal{L}_{sim}$	0.15	0.0001	0.0008	0.0073
			w/o $\mathcal{L}_{KL}$ & $\mathcal{L}_{sim}$	0.07	0.0002	0.0041	0.0107
Shelf	0.0361	5.03	Baseline		0.0260		
			Simple		0.0008		
			Ours	0.25	0.0006	0.0031	0.0063
			w/o $\mathcal{L}_{KL}$	0.22	0.0025	0.0235	0.0231
			w/o $\mathcal{L}_{sim}$	0.41	0.0014	0.0047	0.0073
			w/o $\mathcal{L}_{KL}$ & $\mathcal{L}_{sim}$	0.24	0.0016	0.0173	0.0268
Bed	0.0136	10.17	Baseline		0.0060		
			Simple		0.0003		
			Ours	0.23	0.0004	0.0015	0.0109
			w/o $\mathcal{L}_{KL}$	0.13	0.0006	0.0044	0.0078
			w/o $\mathcal{L}_{sim}$	0.15	0.0004	0.0016	0.0119
			w/o $\mathcal{L}_{KL}$ & $\mathcal{L}_{sim}$	0.12	0.0006	0.0047	0.0075

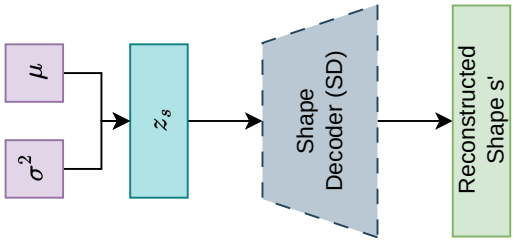

Figure 8: *Network components involved in parameter prediction via optimization. Mainly the shape decoder SD is required.*

sual examples that shapes reconstructed with our network via optimization are again quite close to the original shapes, with differences only in some localized features. The average Chamfer distances (D_{chamfer}) between the output meshes and the ground truth meshes are reported in the 7th column of Table 1. We note that the Chamfer distances are slightly higher than those provided by the network in Task 1, but still in acceptable ranges. This is reasonable since we are not relying on the learned mapping but searching it via optimization, which implies that we may not converge to the learned optimum. However, the results are still competitive, especially when comparing to the baseline. The average time taken for parameter prediction using this approach is 1.7s per shape. Although the inference time is higher than the previous approach, it is also still on par with that of the baseline method. This indicates that optimization for inverse mapping also works satisfactorily and we

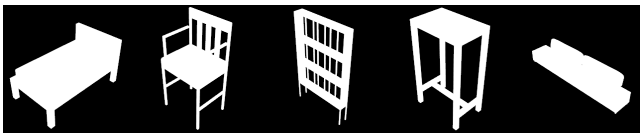

Figure 9: *Examples of silhouette images from which shapes are reconstructed.*

can use it to enable a variety of other tasks. We explore one such task in the next section.

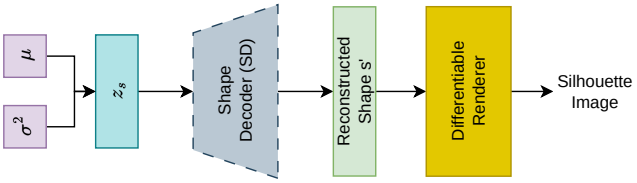

Figure 10: *Network components used in the parameter prediction from silhouette images via optimization.*

4.2.4. Task 4: Shape Reconstruction from Silhouette Images

In this experiment, we reconstruct shapes from their silhouette images. Figure 9 shows example images used in this experiment.

To reconstruct shapes from silhouette images, we use the shape decoder SD as described in Section 4.2.3 with the additional modification that the output of the shape decoder is rendered into silhouette images by a differentiable renderer [LB14, RRN*20], as illustrated in Figure 10. The target can be a single image or an image-stack consisting of multiple views of the same object. The differentiable renderer can be set up with matching number of cameras and their poses, each rendering a different view. The output is a stack of rendered images. In our experiment, we set the number of views $n_v = 4$. Then, we calulate the L_2 loss between the output images and the ground truth images and backpropagate to update μ and σ^2. Once the loss has been minimized, the shapes are reconstructed by decoding μ with the parameter decoder PD and then using the procedural model with the decoded parameters. The last columns in Figure 5 and Figure 6 show examples of shapes reconstructed from silhouettes of the original shapes, while Table 1 reports the average Chamfer distances (D_{chamfer}) between the output meshes and the ground truth meshes for all the test shapes.

We observe that the reconstructions are not as accurate as reconstructions from 3D shapes. However, the overall shape of the objects is still recovered from the little information provided by the silhouettes. The Chamfer distances are also still relatively low (overall below 0.03), implying that the general shapes are recovered. The average time taken for parameter prediction using this approach is 10s per shape. The increase in inference time is primarily due to the network complexity added by the differentiable renderer. This shows the potential of the network for reconstruction from other modalities of data, which we explore more in Section 4.3.

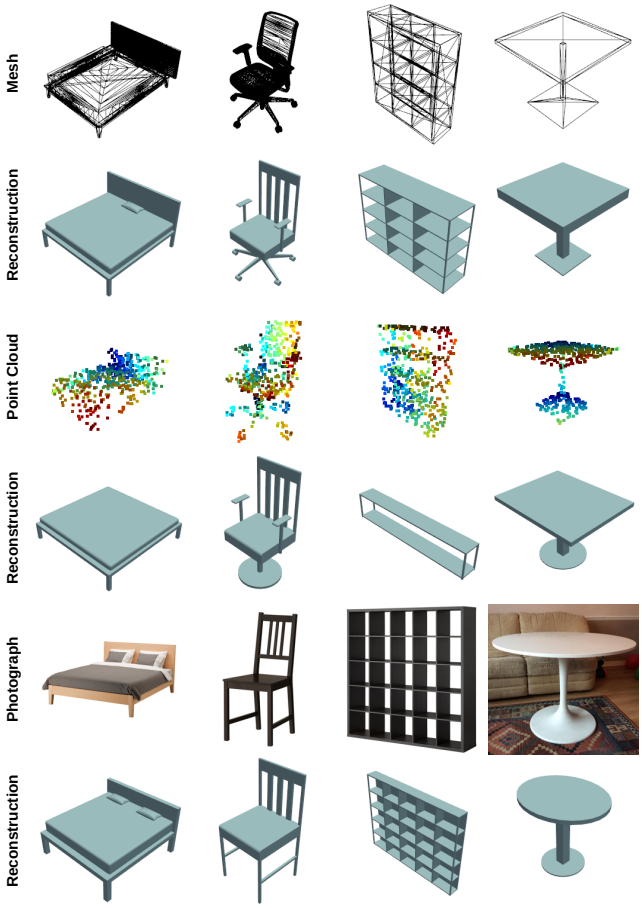

Figure 11: *Reconstructions of shapes from ShapeNet objects (in mesh or point cloud format) and Pix3D images using our method.*

4.3. Applications Enabled by Optimization

Our neural approximation of procedural models is applicable to a diverse range of applications and fares reasonably well with data that was not generated using the same procedural models. To demonstrate this, we reconstruct shapes from various forms of input data, such as meshes, point clouds, and single-view photographs, which are obtained from publicly available sources. In the first experiment, we train our network with data generated by the procedural models described before, and as discussed in Section 4.2.2, we use the trained network to predict parameters for example shapes taken from the ShapeNet dataset [CFG*15]. We selected 10 shapes for each of the bed, chair, shelf, and table classes, respectively. The procedural models then use the predicted parameters to reconstruct the shapes. We perform this experiment on both meshes and point clouds extracted from meshes, both of which can be transformed into voxel-grids and then used by the network. In the second experiment, to reconstruct shapes from photographs, we extract the silhouette of the object we would like to reconstruct and use the method described in Section 4.2.4 to infer the parameters of the shape. Specifically, for each object, we use its photograph and its object mask provided in the Pix3D dataset [SWZ*18] and only a

Table 2: *Chamfer distance between ShapeNet and Pix3D shapes and their reconstructions using different types of input.*

Input type \ Shape class	Bed	Chair	Shelf	Table
Mesh	0.0023	0.0021	0.0079	0.0030
Point cloud	0.0106	0.0057	0.0593	0.0134
Photograph	0.0072	0.0452	0.0129	0.0055

single image is used ($n_v = 1$). After obtaining the parameters, we use the procedural models to generate the reconstructed shapes.

Figure 11 shows selected results from this experiment. Table 2 shows Chamfer distances between the original shapes and their reconstructions. The first column in Table 2 represents input type and the first row shows the class of shapes. In most cases, the reconstructions are fairly close. There are some failure cases, for example, the sparsity of point clouds can lead to inaccuracies in the prediction of fine structures such as the inside of shelves or the chair backs. On the other hand, more standard shapes such as the round table and chair given in the photographs are reconstructed well.

One limitation with reconstructing shapes from images is that the reconstruction quality is heavily dependent on the number of images and the camera poses used by the images. If the right camera poses are available, only a handful of images can lead to good reconstruction. However, this is usually not the case. On the other hand, one can expect a better reconstruction when a large number of images with more variation in camera poses are available. A good example of such a case is when the target shape is available via a novel-view-synthesis method, such as a Neural Radiance Field (NERF) or a 3D Gaussian Splatting (3DGS) scene. With such representation, an arbitrary number of images of the target shape can be rendered with varying camera poses, which can yield a better reconstruction using our network.

To demonstrate this idea, we show how our method can be used to reconstruct shapes from 3DGS scenes. For this experiment, we record a video by moving a physical camera around a real chair with the camera always pointed towards the chair. We then sample frames from the video and construct a 3DGS representation of the scene from the sampled frames. We select a set of 50 camera poses that are not used during the training process and render the 3DGS scene using the camera poses. From the rendered images, the silhouette of the chair is extracted using pretrained YOLO (You Only Look Once) and SAM (Segment Anything) models [Joc20, KMR*23]. The silhouette images are then used to reconstruct the 3D shape with the same process described in Section 4.2.4. However, we use 50 images instead of 4 ($n_v = 50$). Even though in this particular instance we could directly sample these images from the video, constructing a 3DGS scene from them allows us to synthesize views that are not readily available from the video. Figure 12 shows the results from this experiment. The top row of images on the left are images rendered from the 3DGS and the bottom row shows the silhouettes extracted from the corresponding images. Several views of the reconstructed shape are shown on the right.

Table 3: *Average Chamfer distance between original shapes and their reconstructions for ShapeAssembly and our method. We highlight the best result for each shape class.*

Method \ Shape class	Chair	Shelf	Table
ShapeAssembly	0.0068	0.0048	0.0071
Ours	0.0081	0.0187	0.0065

4.4. Comparison with ShapeAssembly

To provide a comparison to state-of-the-art methods aimed at reconstructing 3D shapes in a procedural manner, we compare shapes reconstructed by our method and ShapeAssembly [JBX*20]. For this comparison, we use the chair, shelf and table classes in the ShapeNet dataset of 3D shapes. In order to reconstruct ShapeNet shapes with ShapeAssembly, the original triangle-mesh shapes are transformed into point clouds first. The point clouds are encoded into latent vectors using the point cloud encoder and then decoded into shape generating programs using the pre-trained decoder. The programs are subsequently parsed into triangle meshes. It is worth mentioning here that the pre-trained point cloud encoder is no longer compatible with the latest version of the code, which is why it is retrained and the results may vary from the ones shown in the original paper. With our method, the original shapes are voxelized and encoded into latent vectors, and then decoded into parameter vectors. The procedural model then generates the shapes from the parameter vectors. For this experiment, we reconstructed 5617, 1768 and 7654 shapes for chair, shelf and table shapes respectively, using both methods. Figure 13 shows some of the reconstructed shapes. The top row shows the original shapes. The second and the third rows show shapes reconstructed by ShapeAssembly and our method, respectively. Table 3 shows the quality of the reconstructions for all the shapes in the dataset.

The results show that ShapeAssembly performs better on the shelf category, while the two methods are comparable on the other two categories. We see in the visual results that both methods can provide good reconstructions for certain shapes. ShapeAssembly has the advantage of providing a method for automatically extracting shape programs from a dataset and predicting programs for 3D shapes, while our method allows any procedural model to be used in an agnostic manner, and can perform inverse mapping via optimization, enabling inverse mapping from partial views (images). One limitation of ShapeAssembly is that it is possible for the network to generate shape programs that result in implausible geometry, illustrated by the second chair example. Our method on the other hand ensures valid reconstructions since the predicted parameters are provided to a procedural model.

4.5. Ablation Study

We conduct an ablation study where we investigate the effect of regularizing the latent vectors. Regularization on the latent space is done using two separate techniques. We sample the latent vector z from a normal distribution and we penalize the network if similar parameter vectors are encoded to dissimilar latent vectors.

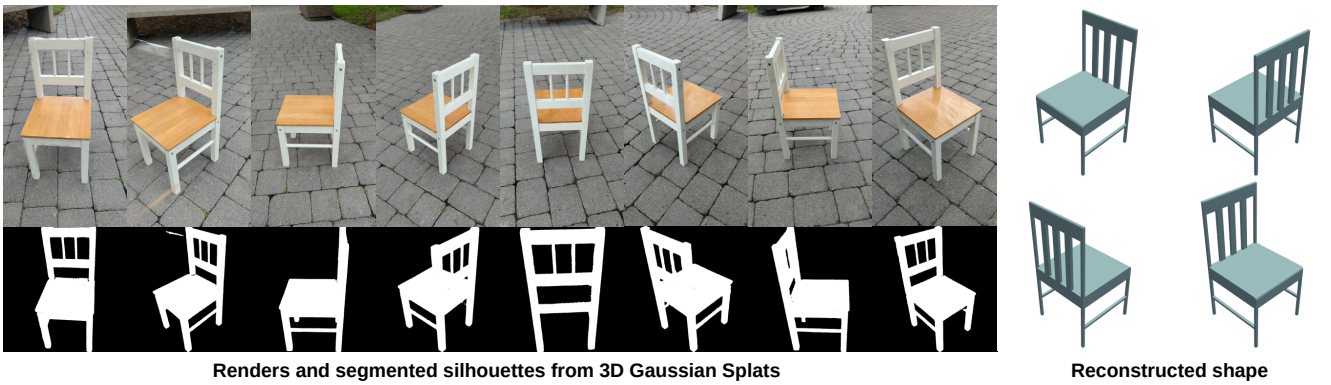

Figure 12: *Results showing how our method can reconstruct shapes with a procedural model from 3D Gaussian Splats.*

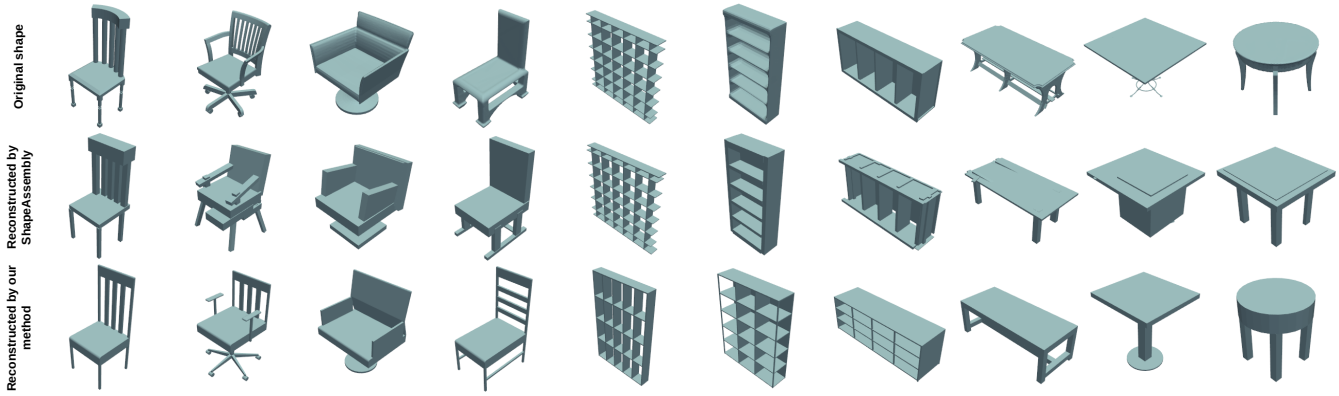

Figure 13: *Comparison between shapes reconstructed by ShapeAssembly and our method on selected shapes.*

In this experiment, we study their effect by removing the restrictions, both individually and combined. In other words, we train one network where z is deterministic and the term $\mathcal{L}_{KL}$ is omitted from the loss function ("w/o $\mathcal{L}_{KL}$" in Table 1), and another network where the term $\mathcal{L}_{sim}$ from the loss function is omitted ("w/o $\mathcal{L}_{sim}$" in Table 1). We train yet another network with both of these modifications ("w/o $\mathcal{L}_{KL}$ & $\mathcal{L}_{sim}$" in Table 1). We repeat the above mentioned experiments with these modified networks and measure which network performs better. Table 1 summarizes the findings, which indicate that regularizing the latent space results in overall better performance for all the tasks.

5. Limitations

There are some limitations to the approach taken in this work. Care needs to be taken when designing the procedural model, so that the output shapes do not suffer from bad geometry. One way to ensure this is by specifying manual constraints as part of the procedural model. However, this can affect the performance of the neural network representation of the procedural models. Another limitation is that voxels do not capture fine details very well. This restricts the applicability of our method to procedural models where the output shapes do not have intricate details. Lastly, procedural models with large numbers of parameters pose a challenge to our approach, since the neural network needs to be trained on a large dataset, which requires more computation.

6. Conclusions and future work

We introduced a neural network architecture to approximate procedural models of 3D shapes. We demonstrated that the network is able to learn the forward and inverse mappings of procedural models implemented in different manners, making the network implementation-agnostic, and can also be used in optimization-based inversion tasks such as reconstruction from images.

In future work, we would like to evaluate our method for approximating procedural models of organic shapes such as vegetation, which tend to have finer multi-scale features. Thus, we would also like to investigate whether other shape representations can be used to better capture fine details of shapes, and explore how our approach scales with the number of parameters. In addition, we would like to experiment with using stochastic gradient estimates for inverse procedural modeling [DHB24]. Finally, although our current method focuses on predicting parameters of the procedural model, it would be an exciting direction if we can further align our latent space with the CLIP [RKH*21] latent space in order to also offer the flexibility to the user of manipulating the parameters and shapes through textual inputs.

References

[Bas21] BASH B.: Procedural furniture chair - sofa - table - bed, 2021. URL: https://blenderbash.gumroad.com/l/tPJOa?layout=profile. 6

[BWS10] BOKELOH M., WAND M., SEIDEL H.-P.: A connection between partial symmetry and inverse procedural modeling. *ACM Trans. on Graphics 29*, 4 (2010), 104:1–10. 3

[CFG*15] CHANG A. X., FUNKHOUSER T., GUIBAS L., HANRAHAN P., HUANG Q., LI Z., SAVARESE S., SAVVA M., SONG S., SU H., XIAO J., YI L., YU F.: *ShapeNet: An Information-Rich 3D Model Repository*. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 11

[CSQ*22] CASCAVAL D., SHALAH M., QUINN P., BODIK R., AGRAWALA M., SCHULZ A.: Differentiable 3d cad programs for bidirectional editing. *Computer Graphics Forum 41*, 2 (2022), 309–323. 2, 3

[DAB14] DEMIR I., ALIAGA D. G., BENES B.: Proceduralization of buildings at city scale. In *Proc. 3D Vision (3DV)* (2014), vol. 1, pp. 456–463. 3

[DAB16] DEMIR I., ALIAGA D. G., BENES B.: Proceduralization for editing 3d architectural models. In *Proc. 3D Vision (3DV)* (2016), pp. 194–202. 3

[DHB24] DELIOT T., HEITZ E., BELCOUR L.: Transforming a non-differentiable rasterizer into a differentiable one with stochastic gradient estimation. *Proc. ACM Comput. Graph. Interact. Tech. 7*, 1 (May 2024). 13

[GDGP16] GUÉRIN E., DIGNE J., GALIN E., PEYTAVIE A.: Sparse representation of terrains for procedural modeling. *Computer Graphics Forum 35*, 2 (2016), 177–187. 3

[GHS*22] GUERRERO P., HAŠAN M., SUNKAVALLI K., MĚCH R., BOUBEKEUR T., MITRA N. J.: Matformer: a generative model for procedural materials. *ACM Trans. on Graphics 41*, 4 (2022), 46:1–12. 3

[GJB*20] GUO J., JIANG H., BENES B., DEUSSEN O., ZHANG X., LISCHINSKI D., HUANG H.: Inverse procedural modeling of branching structures by inferring l-systems. *ACM Trans. on Graphics 39*, 5 (2020). 3

[GKG*22] GAILLARD M., KRS V., GORI G., MĚCH R., BENES B.: Automatic differentiable procedural modeling. *Computer Graphics Forum 41*, 2 (2022), 289–307. 2, 3

[HDR19] HU Y., DORSEY J., RUSHMEIER H.: A novel framework for inverse procedural texture modeling. *ACM Trans. on Graphics 38*, 6 (2019), 186:1–14. 3

[HGH*22] HU Y., GUERRERO P., HASAN M., RUSHMEIER H., DESCHAINTRE V.: Node graph optimization using differentiable proxies. In *Proc. SIGGRAPH* (2022), pp. 5:1–9. 3

[HGH*23] HU Y., GUERRERO P., HASAN M., RUSHMEIER H., DESCHAINTRE V.: Generating Procedural Materials from Text or Image Prompts. In *Proc. SIGGRAPH* (2023), pp. 4:1–11. 3

[HHD*22] HU Y., HE C., DESCHAINTRE V., DORSEY J., RUSHMEIER H.: An inverse procedural modeling pipeline for svbrdf maps. *ACM Trans. on Graphics 41*, 2 (2022), 18:1–17. 3

[HHL*24] HU J., HUI K.-H., LIU Z., LI R., FU C.-W.: Neural wavelet-domain diffusion for 3d shape generation, inversion, and manipulation. *ACM Trans. on Graphics 43*, 2 (2024), 16:1–18. 3

[HKYM17] HUANG H., KALOGERAKIS E., YUMER E., MECH R.: Shape synthesis from sketches via procedural models and convolutional networks. *IEEE Trans. Visualization & Computer Graphics 23*, 8 (2017), 2003–2013. 2, 3

[HLHF22] HUI K.-H., LI R., HU J., FU C.-W.: Neural wavelet-domain diffusion for 3d shape generation. In *Proc. SIGGRAPH Asia* (2022), pp. 24:1–9. 2

[HPG*22] HERTZ A., PEREL O., GIRYES R., SORKINE-HORNUNG O., COHEN-OR D.: SPAGHETTI: editing implicit shapes through part aware generation. *ACM Trans. on Graphics 41*, 4 (2022), 106:1–20. 2

[JBX*20] JONES R. K., BARTON T., XU X., WANG K., JIANG E., GUERRERO P., MITRA N. J., RITCHIE D.: ShapeAssembly: learning to generate programs for 3D shape structure synthesis. *ACM Trans. on Graphics (Proc. SIGGRAPH Asia) 39*, 6 (2020), 234:1–10. 3, 12

[JCG*21] JONES R. K., CHARATAN D., GUERRERO P., MITRA N. J., RITCHIE D.: Shapemod: Macro operation discovery for 3d shape programs. *ACM Trans. on Graphics (Proc. SIGGRAPH) 40*, 4 (2021), 153:1–16. 3

[JGMR23] JONES R. K., GUERRERO P., MITRA N. J., RITCHIE D.: Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Trans. on Graphics 42*, 4 (2023), 49:1–17. 3

[Joc20] JOCHER G.: Ultralytics yolov5, 2020. URL: https://github.com/ultralytics/yolov5, doi:10.5281/zenodo.3908559. 12

[JWR22] JONES R. K., WALKE H., RITCHIE D.: Plad: Learning to infer shape programs with pseudo-labels and approximate distributions. *Proc. IEEE Conf. on Computer Vision & Pattern Recognition* (2022), 9871–9880. 3

[KGTK24] KUSUPATI U., GAILLARD M., THIERY J.-M., KAISER A.: Semantic shape editing with parametric implicit templates. In *Proc. SIGGRAPH* (2024). 3

[KMG*21] KRS V., MĚCH R., GAILLARD M., CARR N., BENES B.: PICO: procedural iterative constrained optimizer for geometric modeling. *IEEE Trans. Visualization & Computer Graphics 27*, 10 (2021), 3968–3981. 2

[KMR*23] KIRILLOV A., MINTUN E., RAVI N., MAO H., ROLLAND C., GUSTAFSON L., XIAO T., WHITEHEAD S., BERG A. C., LO W.-Y., DOLLÃ¡R P., GIRSHICK R.: Segment anything, 2023. arXiv:2304.02643. 12

[LB14] LOPER M. M., BLACK M. J.: Opendr: An approximate differentiable renderer. In *Proc. Euro. Conf. on Computer Vision* (2014), pp. 154–169. 11

[LGT*23] LIN C.-H., GAO J., TANG L., TAKIKAWA T., ZENG X., HUANG X., KREIS K., FIDLER S., LIU M.-Y., LIN T.-Y.: Magic3d: High-resolution text-to-3d content creation. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition* (2023), pp. 300–309. 2

[LMW*22] LIN C., MITRA N., WETZSTEIN G., GUIBAS L. J., GUERRERO P.: Neuform: Adaptive overfitting for neural shape editing. In *Advances in Neural Information Processing Systems (NeurIPS)* (2022), vol. 35, pp. 15217–15229. 3

[LSL*19] LIPP M., SPECHT M., LAU C., WONKA P., MÜLLER P.: Local editing of procedural models. *Computer Graphics Forum 38*, 2 (2019), 13–25. 2, 3

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. In *Proc. SIGGRAPH* (2008), pp. 102:1–10. 2

[MB21] MICHEL E., BOUBEKEUR T.: Dag amendment for inverse control of parametric shapes. *ACM Trans. on Graphics 40*, 4 (2021), 173:1–14. 2, 3

[Mer23] MERRELL P.: Example-based procedural modeling using graph grammars. *ACM Trans. on Graphics 42*, 4 (2023), 60:1–16. 3

[MM11] MERRELL P., MANOCHA D.: Model synthesis: A general procedural modeling algorithm. *IEEE Trans. Visualization & Computer Graphics 17*, 6 (2011), 715–728. 2

[MPZ20] MATHUR A., PIRRON M., ZUFFEREY D.: Interactive programming for parametric cad. *Computer Graphics Forum 39*, 6 (2020), 408–425. 3

[NBA18] NISHIDA G., BOUSSEAU A., ALIAGA D. G.: Procedural modeling of a building from a single image. *Computer Graphics Forum (Proc. Eurographics) 37*, 2 (2018), 415–429. 3

[NGDA*16] NISHIDA G., GARCIA-DORADO I., ALIAGA D. G., BENES B., BOUSSEAU A.: Interactive sketching of urban procedural models. *ACM Trans. on Graphics 35*, 4 (2016), 130:1–11. 2, 3

[Pat12] PATOW G.: User-friendly graph editing for procedural modeling of buildings. *IEEE Computer Graphics and Applications 32*, 2 (2012), 66–75. 2

[PLH*22] PEARL O., LANG I., HU Y., YEH R. A., HANOCKA R.: Geocode: Interpretable shape programs. 3

[Rag22] RAGONNEAU T. M.: *Model-Based Derivative-Free Optimization Methods and Software*. PhD thesis, Department of Applied Mathematics, The Hong Kong Polytechnic University, Hong Kong, China, 2022. URL: https://theses.lib.polyu.edu.hk/handle/200/12294. 6

[RKH*21] RADFORD A., KIM J. W., HALLACY C., RAMESH A., GOH G., AGARWAL S., SASTRY G., ASKELL A., MISHKIN P., CLARK J., ET AL.: Learning transferable visual models from natural language supervision. In *Proc. Int. Conf. on Machine Learning* (2021), pp. 8748–8763. URL: https://proceedings.mlr.press/v139/radford21a/radford21a.pdf. 13

[RRN*20] RAVI N., REIZENSTEIN J., NOVOTNY D., GORDON T., LO W.-Y., JOHNSON J., GKIOXARI G.: Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501* (2020). 11

[RZ24] RAGONNEAU T. M., ZHANG Z.: COBYQA Version 1.1.2, 2024. URL: https://www.cobyqa.com. 6

[ŠBM*10] ŠTAVA O., BENES B., MĚCH R., ALIAGA D. G., KRIŠTOF P.: Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum 29*, 2 (2010), 665–674. 3

[SGL*22] SHARMA G., GOYAL R., LIU D., KALOGERAKIS E., MAJI S.: Neural shape parsers for constructive solid geometry. *IEEE Trans. Pattern Analysis & Machine Intelligence 44*, 5 (2022), 2628–2640. 3

[SLH*20] SHI L., LI B., HAŠAN M., SUNKAVALLI K., BOUBEKEUR T., MECH R., MATUSIK W.: Match: Differentiable material graphs for procedural material capture. *ACM Trans. on Graphics 39*, 6 (2020), 196:1–15. 3

[SPK*14] STAVA O., PIRK S., KRATT J., CHEN B., MĚCH R., DEUSSEN O., BENES B.: Inverse procedural modelling of trees. *Computer Graphics Forum 33*, 6 (2014), 118–131. 2, 3

[SSW*24] SHEN I., SU L.-W., WU Y.-T., CHEN B.-Y., ET AL.: Stylepart: image-based shape part manipulation. *The Visual Computer* (2024), 1–12. 3

[STBB14] SMELIK R. M., TUTENEL T., BIDARRA R., BENES B.: A survey on procedural modelling for virtual worlds. *Computer Graphics Forum 33*, 6 (2014), 31–50. 2

[SWZ*18] SUN X., WU J., ZHANG X., ZHANG Z., ZHANG C., XUE T., TENENBAUM J. B., FREEMAN W. T.: Pix3d: Dataset and methods for single-image 3d shape modeling. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition* (2018). 11

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM Trans. on Graphics 30*, 2 (2011), 11:1–14. 2

[TLS*19] TIAN Y., LUO A., SUN X., ELLIS K., FREEMAN W. T., TENENBAUM J. B., WU J.: Learning to infer and execute 3d shape programs. In *Proc. Int. Conf. on Learning Representations* (2019), pp. 1–21. 3

[TMK*19] TRUNZ E., MERZBACH S., KLEIN J., SCHULZE T., WEINMANN M., KLEIN R.: Inverse procedural modeling of knitwear. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition* (2019), pp. 8622–8631. 3

[WYD*14] WU F., YAN D.-M., DONG W., ZHANG X., WONKA P.: Inverse procedural modeling of facade layouts. *ACM Trans. on Graphics 33*, 4 (2014), 121:1–10. 3

[YAMK15] YUMER M. E., ASENTE P., MĚCH R., KARA L. B.: Procedural modeling using autoencoder networks. In *Proc. Symp. on User Interface Software and Technology* (2015), pp. 109–118. 3

[YBP*24] YUAN H., BOUSSEAU A., PAN H., ZHANG Q., MITRA N. J., LI C.: Diffcsg: Differentiable csg via rasterization. In *Proc. SIGGRAPH Asia* (2024). 3

[YLM*22] YAN X., LIN L., MITRA N. J., LISCHINSKI D., COHEN-OR D., HUANG H.: Shapeformer: Transformer-based shape completion via sparse representation. In *Proc. IEEE Conf. on Computer Vision & Pattern Recognition* (2022), pp. 6239–6249. 2

[ZTNW23] ZHANG B., TANG J., NIESSNER M., WONKA P.: 3dshape2vecset: A 3d shape representation for neural fields and generative diffusion models. *ACM Trans. on Graphics 42*, 4 (2023). 2

[ZVW*22] ZENG X., VAHDAT A., WILLIAMS F., GOJCIC Z., LITANY O., FIDLER S., KREIS K.: Lion: Latent point diffusion models for 3d shape generation. In *Advances in Neural Information Processing Systems (NeurIPS)* (2022), vol. 35, pp. 10021–10039. 2, 3